Perl Lesson 1

Hello World!

What you will learn

- What perl is and what it is good for
- How to create a perl script in Linux
- How to run a perl script in Linux
- Output to the Screen
- Scalar Variables
- Dealing with User Input
- Comparison Operators and when to use them
- Exercises/Homework

What perl is

What perl is and what it is good for

Practical Extraction and Report Language?

Pathologically Eclectic Rubbish Lister?

Larry Wall, the Father of Perl, will agree to both of these acronyms. Perl is an excellent parser, web page, database interface, report writer, e-mailer, shell script, daemon, OS interface, security and encryption tool, image manipulator, file handler, programming language interface(wait, what does that even mean?), and most of all it is still the glue of the Internet!

Oh, did I mention it runs IMARS?

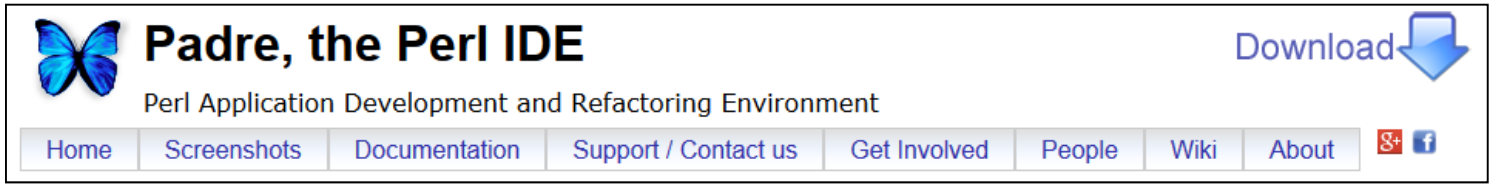
How to install perl in Windows

How to install perl in Windows

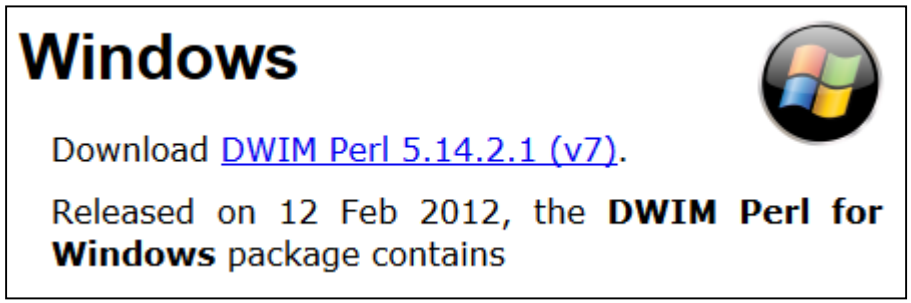
Open your browsers and goto:

- http://padre.perlide.org/

Click on “Download” in the upper right corner:



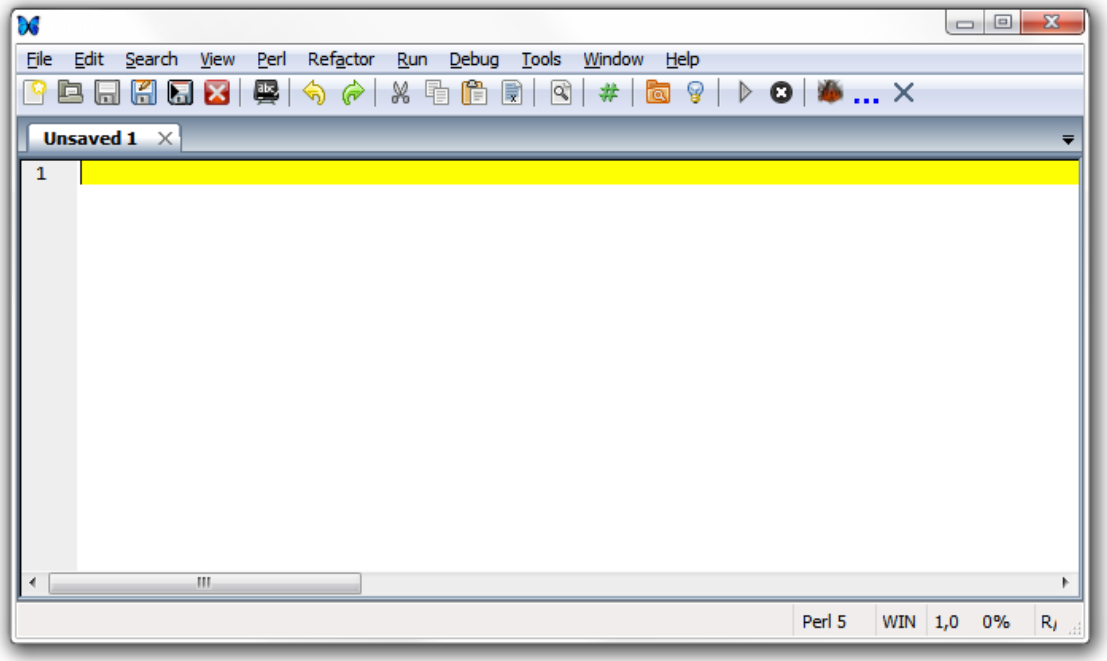
Now click on “DWIM Perl 5.14.2.1 (v7)”:



After the download is complete, please install the software package.

How to install perl in Windows

Now that you have installed perl, please open Parde.



Parde is a perl IDE and is where most of this class will take place. I figured that it would be easier to teach perl in Windows than it would be to try and have everyone use vi on Linux.

How to install perl in Windows

Now we are going to write your first perl script! You should have Parde open and there should be a window to type in. On the first line please add the following line of code:

```
#!/usr/bin/perl
```

This line of code simply lets the OS know that it's dealing with a perl script and where to find perl.

Now let's save the file as my_first_perl.pl, you should be able to figure out how to do this. After you have finished saving the perl script we need to run it. In Parde the default "run" hotkey is F5. Please hit F5 now! You should now see this window

How to create a perl script in Linux

How to create a perl script in Linux

A perl script is a simple text file that gets complied at run time. This means there is no need for any fancy UI or text editor. I only code perl in vi.

Command for creating a perl script:

```
vi my_first_perl.pl
```

Now we need to add the below string to the beginning of the file. This lets Linux know that it's supposed to interrupt the file as a perl script.

```
#!/usr/bin/perl
```

Please save the file and return to a Linux Prompt. Now we need to make the script executable. We do this with chmod:

```
chmod a+x my_first_perl.pl
```

Now lets run it.

```
./my_first_perl.pl
```

The script should just return a blank line and give you the Linux prompt again.

Output to the Screen

Output to the Screen

Now that we have created at least one script it's time to start learning how to control perl. Lets try a Hello World script! Please input the following into the Parde window.

```
#!/usr/bin/perl

print "Hello World!\n";
```

Now that you have added the code to your script please hit F5 to run it. Do you see the below window?

Output to the Screen

Lets breakdown the different parts of what you just programmed. We already know that the first line is for letting the OS know what type of file this is, so lets skip it.

There are 5 parts to the below command.

```
print “Hello World!\n”;
```

1. print, This lets perl know that you are trying to output something to the user. This could be printing to the Screen(STDOUT), a file, or even to a device like a printer.
2. Quotation Marks, These let perl know what you want to output.
3. Hello World!, This is simple. This is what gets output.
4. \n, This lets perl know that you would like a newline after the output.
5. Semi-colon, This is how perl knows that it needs to stop expecting information to output. Really this is how all commands in most programming languages are completed.

Output to the Screen

Lets go into \n a little deeper.

\n, This represents a new line however there are two parts to this command.

1. \, This is a back slash. It's also considered the escape character in perl. This means it will allow a special function to be followed based on the next character. It can also be used to stop perl from interpreting normal syntax, such as a Quotation Mark.

2. n, This lets perl know what you want to do now that you have escaped.

   - n, is used to output a newline
   - t, is used to output a tab
   - There are many others, mostly used for regex(don't worry about regex that will be covered in a later lesson)

Variables

Variables

Variables are containers that store sections of data.

Some common variable types in programming are:

1. Integer, This is a whole number. ‘1’, ’10’, ’100’
2. Floating Point, This is a real number, or a number with a decimal point.
3. Boolean, True/False, On/Off, 1/0
4. String, This holds a word or a sentence
5. Array, This is a Variable that hold more than one Variable and is indexed on a numeric scheme
6. Multi Dimensionally Array, This is a variable that holds more than one Array and is indexed on a numeric scheme
7. Associative Array, This is an array that’s index is defined by the program it’s self, it doesn’t use numbers for an index. But it could if you wanted.

Variables

Perl is pretty unique as to how it deals with Variables. Here are the three variable types that perl uses: Scalar, Array, and Hashes.

Scalar Variables are used by perl to cover the following variable types: Integer, Floating Point, Boolean, and String.

```
$var = “variable”;

$firstname = “Michael”;

$email = “mstpehens@csid.com”;
```

Arrays in perl are just like most other languages.

```
@vars = (“variable1”, “variable2”, “variable3”);

@firstnames = (“Michael”, “David”, “Adrian”);

@emails = (mstpehens@csid.com, jross@csid.com, robot@lifelock.com)
```

Hashes are used by perl to cover Associative Arrays(Hashes are pretty powerful, lets cover these in their own lesson)

Variables

As I said perl is pretty unique as to how it deals with Variables. Here are somethings that make it unique:

- No need to declare Variables before they are used
- No declaration of Variable Types; not counting Scalar, Array, or Hash
- Variable Types are identifiyed by their cast character; $ = Scalar, @ = Array, and % = Hashes
- There is no such thing as an Integer, not really anyway
- There is no such thing as a Boolean, not really anyway
- Scalars can convert Integers, to Floating Points, to Booleans, to Strings, and back again. This assumes that a certain degree of data loss is ok. For example, a Floating Point Variable would loose any decimal points if converted to an Integer but, it would keep all of the decimal points if it was converted to a String.

Variables

Lets play around with some Scalar Variables. Please enter the following code and hit F5 to run it.

```
$var = “Hello World!”;

print $var;
```

What happened? Did it print out “Hello World!”? Is it missing anything?

You might have noticed that it printed “Hello World!” to one line and then the text “Press any key to continue…” was printed right after the exclamation point. This is due to the lack of a \n, or newline. Lets add the newline to the code and hit F5.

```
$var = “Hello World!”;

print $var\n;
```

Did it print what you expected? What is wrong with this code and how can we fix it?

Variables

Lets fix the code, here are two different ways to fix it.

We can add the \n to $var.

```
$var = “Hello World!\n”;

print $var;
```

Or we can enclose the variable in Quotation Marks and add \n to the print statement.

```
$var = “Hello World!”;

print “$var\n”;
```

Both of these examples will print the same thing. Please try both now.

Dealing with User Input

Dealing with User Input

I hardly ever write perl scripts that will take user input that isn't on the command line but, it's a quick and easy method of changing variables with each run. So lets learn about Standard In <STDIN>. Standard In is a method of taking input. When <STDIN> is called in perl the script will pause until a carriage return(Enter) is entered. Please type in the following code and hit F5:

```
print "Who is the best boss in the world?";

$var = <STDIN>;

print "$var is the best boss in the world!\n";
```

Did that come out as you expected? What two things should be fixed and how do we fix them?

```
Who is the best boss in the world?Ken
Ken
 is the best boss in the world!
Press any key to continue . . .
```

Dealing with User Input

The two things that I see wrong with this script are:

1. There should be a space in between the Question Mark and where we start typing our answer.

```
print “Who is the best boss in the world?”;
```

Becomes

```
print “Who is the best boss in the world? ”;
```

2. Our answer is on a line all by itself. This is because <STDIN> will include the carriage return. To deal with this we need to learn a new command called chomp. Chomp will remove the last character of a variable if it is a \n. It’s simple to use this command as well. Try adding this to your code and then hit F5

```
$var = <STDIN>;

chomp $var;

print “\n$var is the best boss in the world!\n”;
```

Comparison Operators and when to use them

Comparison Operators and When to Use Them

Comparison Operators are how perl is able to apply your logic in the script. These are things like equal, not equal, greater than, less than, greater than or equal, and less than or equal. Since perl lets you move one variable type into another variable type on the fly there are different Comparison Operators for Numeric and String values. How you call the Operator is how perl will decide if you are looking at a variable as a Number or a String.

Comparison	Numeric	String
Equal	==	eq
Not equal	!=	ne
Less than	<	lt
Greater than	>	gt
Less than or equal	<=	le
Greater than or equal	>=	ge

If Statements

If statements are how perl applies Comparison Operators to make your logic come to life. When an If statement is found to be true then a section of code below it will be executed.

35	>	5	TRUE
35	!=	45	TRUE
35	!=	45 - 10	FALSE
‘35'	eq	‘35.0'	FALSE
‘CSID'	eq	‘CSID'	TRUE
‘CSID'	eq	‘csid'	FALSE

If Statements

If statements are fairly simple to build, use, and to understand. Type the below code in and try to run it.

```
print "Who is the best boss in the world? ";
$var = <STDIN>;
chomp $var;
if ($var eq 'Ken') {
    print "$var is the best boss in the world!\n";
}
```

Now try both of these strings: “Ken” and “ken”.

Do you understand what happened when you entered “ken”.

If Else Statements

Else statements allow code to be run when the If statement isn't found to be true. Lets add the blow code to our script.

```
if (($var eq 'Ken')  or ($var eq 'ken')){
  print "$var is the best boss in the world!\n";
} else {
  print "I don't believe you know what you are talking about\n";
}
```

Now try and run the and enter anything other than "Ken" or "ken".

What you have learned

- What perl is
- How to create a perl script in Linux
- How to run a perl script in Linux
- Output to the Screen
- Scalar Variables
- Dealing with User Input
- Comparison Operators and when to use them

Exercises / Homework

Exercises / Homework

APPLAUSE **Thank You** APPLAUSE